- Windows Priority Scheduling
- In windows, The priority levels range from zero (lowest priority) to 31 (highest priority).
- The system treats all threads with the same priority as equal. The system assigns time slices in a round-robin fashion to all threads with the highest priority.
- If none of these threads are ready to run, the system assigns time slices in a round-robin fashion to all threads with the next highest priority. If a higher-priority thread becomes available to run, the system ceases to execute the lower-priority thread, and assigns a full time slice to the higher-priority thread.

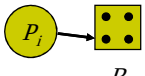
٧٦

Process priority class	Thread priority level	Base p
IDLE_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	2
	THREAD_PRIORITY_BELOW_NORMAL	3
	THREAD_PRIORITY_NORMAL	4
	THREAD_PRIORITY_ABOVE_NORMAL	5
	THREAD_PRIORITY_HIGHEST	6
	THREAD_PRIORITY_TIME_CRITICAL	15
BELOW_NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	4
	THREAD_PRIORITY_BELOW_NORMAL	5
	THREAD_PRIORITY_NORMAL	6
	THREAD_PRIORITY_ABOVE_NORMAL	7
	THREAD_PRIORITY_HIGHEST	8
	THREAD_PRIORITY_TIME_CRITICAL	15
NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	6
	THREAD_PRIORITY_BELOW_NORMAL	7
	THREAD_PRIORITY_NORMAL	8
	THREAD_PRIORITY_ABOVE_NORMAL	9
	THREAD_PRIORITY_HIGHEST	10
	THREAD_PRIORITY_TIME_CRITICAL	15
ABOVE_NORMAL_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1
	THREAD_PRIORITY_LOWEST	8
	THREAD_PRIORITY_BELOW_NORMAL	9
	THREAD_PRIORITY_NORMAL	10
	THREAD_PRIORITY_ABOVE_NORMAL	11
	THREAD_PRIORITY_HIGHEST	12
	THREAD_PRIORITY_TIME_CRITICAL	15
HIGH_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	1

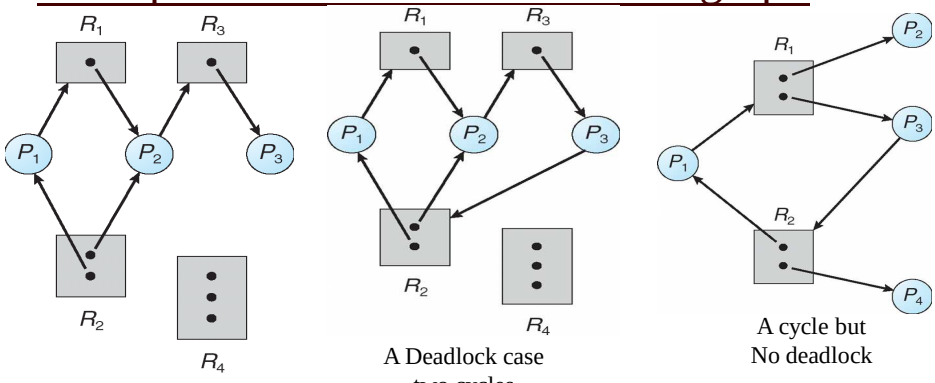
	THREAD_PRIORITY_LOWEST	11
	THREAD_PRIORITY_BELOW_NORMAL	12
	THREAD_PRIORITY_NORMAL	13
	THREAD_PRIORITY_ABOVE_NORMAL	14
	THREAD_PRIORITY_HIGHEST	15
	THREAD_PRIORITY_TIME_CRITICAL	15
REALTIME_PRIORITY_CLASS	THREAD_PRIORITY_IDLE	16
	THREAD_PRIORITY_LOWEST	22
	THREAD_PRIORITY_BELOW_NORMAL	23
	THREAD_PRIORITY_NORMAL	24
	THREAD_PRIORITY_ABOVE_NORMAL	25
	THREAD_PRIORITY_HIGHEST	26
	THREAD_PRIORITY_TIME_CRITICAL	31

Resource-Allocation Graph

- A set of vertices V and a set of edges E.
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.(represented as a circle)
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system . (finite number (4) -represented as rectangle)
  - Every type has a number of identical resource instances
- Request edge – directed edge $P_1 \rightarrow R_j$
- Assignment edge – directed edge $R_j \rightarrow P_i$
  - an instance of (4) resource type Rj has been allocated to process P.
- Each process utilizes a resource as follows:
  - request -use -release



Examples of a resource allocation graph



No cycle

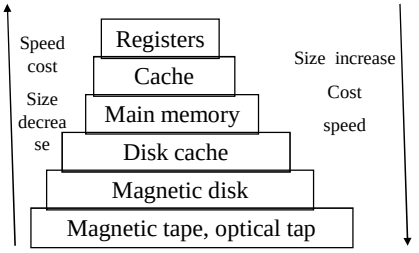
A Deadlock case
two cycles

A cycle but
No deadlock

- If graph contains no cycles ⇒ no deadlock
- If graph contains a cycle :
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

The Memory Hierarchy

- Cache memory: A memory that is smaller and faster than main memory and that is interposed between the processor and the main memory
- Disk cache: a portion of main memory used as a buffer to temporarily hold data that are to be read out to disk.

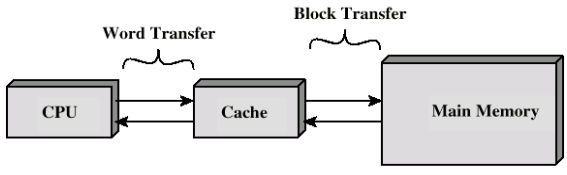


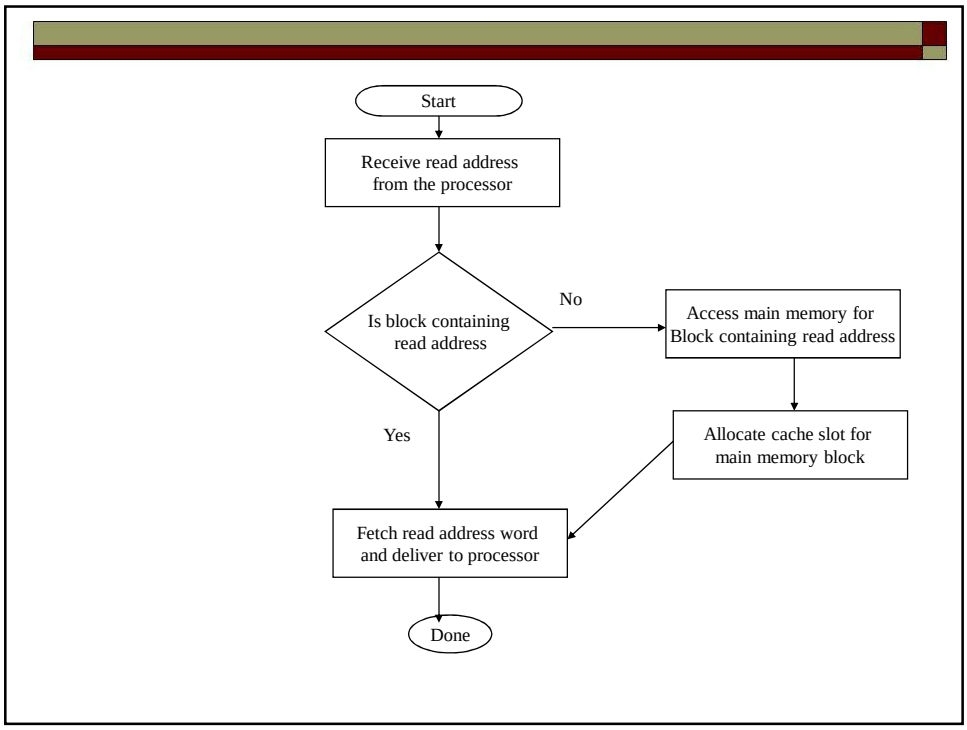
Direct Memory Access (DMA)

- DMA is used by smart high-speed I/O devices able to transmit information at close to memory speeds.
- DMA Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than one interrupt per byte.

Cache memory

- Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time to provide a large memory size
- When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache. If so, the word is delivered to the processor.
- If not, a block of main memory consisting of some fixed number of words is read into cache and then the word is delivered to the processor.





Cache Design

- Cache size : Small caches have a significant impact on performance
- Block size
  - The unit of data exchanged between cache and main memory
  - Larger block size more hits until probability of using newly fetched data becomes less than the probability of reusing data that have to be moved out of cache
- Mapping function
  - Determines which cache location the block will occupy
- Replacement algorithm
  - Determines which block to replace
  - Least-Recently-Used (LRU) algorithm
- Write policy: When the memory write operation takes place
  - Can occur every time block is updated
  - Can occur only when block is replaced
    - Minimizes memory write operations
    - Leaves main memory in an obsolete state

Virtual memory

- Programs usually bigger than main memory, so can't fit into it.
- Virtual memory: A way of expanding main memory of a computer by using disk space for memory functions
  - Access can be hundreds of thousands of times slower than that to main memory

Difference between buffering and caching.

- Buffering: a technique for storing data transmitted from a sending process to a receiving process in local memory or secondary (disk) storage until the receiving process is ready to consume it. (ex, printer buffer)
- Caching: a technique for optimizing access to remote data objects by holding a copy of them in local memory or secondary (disk) storage. Accesses to parts of the remote object are translated into accesses to the corresponding parts of the local copy. Unlike buffering, the local copy may be retained as long as there is local memory available to hold it. (web proxy or cache)

Concurrency & parallelism

Concurrency and parallelism are not the same thing.

Concurrency	Parallelism
Program runs on one single-core.	Program runs on multi-core, multi-processor.
multitasking and asynchronous input-output (I/O)	Multiprocessing
Goal is to reduce latency	Goal is to improve the performance of applications (throughput)
The problem is fundamentally concurrent (OS with multiple tasks)	The problem doesn't inherently require parallel processing